Specification and Verification of a Steam-Boiler with Signal-Coq

Mickael Kerboeuf, David Nowak, Jean-Pierre Talpin

Specification and Verification of a Steam-Boiler with Signal-Coq

Mickaël Kerbœuf[1], David Nowak[2], and Jean-Pierre Talpin[1]

[1]Inria-Rennes – Irisa, Campus de Beaulieu, Rennes, France
[2]Oxford University Computing Laboratory, Wolfson Building, Oxford, England

Abstract Over the last decade, the increasing demand for the validation of safety critical systems has led to the development of domain-specific programming languages (e.g. synchronous languages) and automatic verification tools (e.g. model checkers). Conventionally, the verification of a reactive system is implemented by specifying a discrete model of the system (i.e. a finite-state machine) and then checking this model against temporal properties (e.g. using an automata-based tool). We investigate the use of a synchronous programming language, Signal, and of a proof assistant, Coq, for the specification and the verification of co-inductive properties of the well-known steam-boiler problem.
By way of this large-scale case-study, the Signal-Coq formal approach, i.e. the combined use of Signal and Coq, is demonstrated to be a well-suited and practical approach for the validation of reactive systems. Indeed, the deterministic model of concurrency of Signal, for specifying systems, together with the unparalleled expressive power of the Coq proof assistant, for verifying properties, enables to disregard any compromise incurred by any limitation of either the specification and the verification tools.
Keywords: synchronous programming, theorem proving, the steam-boiler problem.

1 Introduction

In recent years, the verification of safety critical systems has become an area of increasing importance for the development of softwares in sensitive fields: medicine, telecommunication, transportation, energy.

The notion of *reactive system* has emerged to focus on the issues related to the control of interaction and of response-time in mission-critical systems. This has led to the development of specific programming languages and related verification tools for reactive systems.

Conventionally, the verification of a reactive system is implemented by, first, elaborating a *discrete* model of the system (i.e. an approximation of its behaviour by a finite-state machine) specified in a dedicated language (e.g. a synchronous programming language) and, then, by checking a property against the model (i.e. model checking).

Synchronous languages (such as Esterel [5], Lustre [9], Signal [4], Statecharts [10]) have proved to be well adapted to the verification of safety and

liveness properties of reactive systems. For instance, model checking has been used at an industrial scale on SIGNAL programs to check properties such as liveness, invariance, reachability and attractivity.

Whereas model checking efficiently decides discrete properties of finite state systems, the use of formal proof systems enables to prove *numerical and parameterized properties* about *infinite state systems*. Using a proof system, we can not only prove the safety and liveness of a reactive system but also its *correctness* and *completeness*.

Such a proof is of course not automatic and requires interaction with he user to direct its strategy. The prover can nonetheless automate he most tedious and mechanical parts of the proof. In general, formal roofs of programs are difficult and time-consuming. In the very case of modeling a reactive system using a declarative synchronous language, however, this difficulty is milded thanks to the elegant stylistic combination of declarative programming and relational modeling.

We investigate the combined use of the synchronous language SIGNAL and of the proof assistant COQ for specifying and verifying properties of a large-scale case study, namely, the steam-boiler problem.

2 The Signal-Coq Formal Approach

Synchronous languages assume that computation takes no time (this is the so-called "*synchronous hypothesis*"). Actually, this means that the duration of computations is negligible in comparison to the time of reaction of the system. This *synchronous* hypothesis is particularly well adapted to verify safety and some forms of liveness properties. SIGNAL is a synchronous, declarative, data-flow oriented programming language. It is built around a simple paradigm: a process is a system of equations on signals; and a minimal kernel of primitive operators. A signal represents an infinite flow of data. At every instant, it can be absent or present with a value. The instants when values are present are determined by its associated *clock*. Interested reader may find more about SIGNAL in [4].

COQ [7] is a proof assistant for higher-order logic. It allows the development of computer programs that are consistent with their formal specification. The logical language used in COQ is a variety of type theory, the *Calculus of Inductive Constructions* [15]. It has been extended with *co-inductive types* (types defined as greatest fixed points rather than as least fixed points [8]) to handle infinite objects, It is thus well suited to represent signals.

In [14], we have introduced a co-inductive semantics for the kernel of the language SIGNAL and formalized it in the proof assistant COQ. In this section, we summarize the COQ definitions given for the primitive operators of SIGNAL. Interested reader may find the generalization to the complete language in [13].

A signal X is defined as a stream of $\perp$ and values v. Let $\mathcal{D}$ be a set of values. The set of signals $\mathcal{S}_\mathcal{D}$ is the largest set such that:

$$\mathcal{S}_\mathcal{D} = \{\perp.X \mid X \in \mathcal{S}_\mathcal{D}\} \cup \{v.X \mid v \in D, X \in \mathcal{S}_\mathcal{D}\}$$

Instantaneous Relation. The relation R_P^n is used in SIGNAL to specify an instantaneous relation between n signals. At each instant, these signals satisfy the predicate P. In COQ, according to the Curry-Howard isomorphism, a pair proof-specification is represented by a pair term-type. The type of non-well-founded proofs of R_P^n is introduced as a co-inductive type. Co-induction is needed to deal with infinite signals. For instance, R_P^2 is introduced as follows:

```
CoInductive Relation2 [U,V:Set; P:U->V->Prop] :
 (Signal U)->(Signal V)->Prop :=
  relation2_a: (X:(Signal U))(Y:(Signal V))
    (Relation2 P X Y)->
    (Relation2 P (Cons (absent U) X) (Cons (absent V) Y))
| relation2_p: (X:(Signal U))(Y:(Signal V))(u:U)(v:V)
    (P u v)->(Relation2 P X Y)->
    (Relation2 P (Cons (present u) X) (Cons (present v) Y)).
```

Down-Sampling. The SIGNAL equation $Z := X$ **When** Y states that the signal Z down-samples X when X is present and when Y is present with the value *true*. **When** is the least fixpoint of the following continuous functional:

$$F_{\mathbf{When}}(f) =_{\text{def}} \begin{cases} (\bot.X, \bot.Y) \longmapsto \bot.f(X,Y) \\ (\bot.X, b.Y) \longmapsto \bot.f(X,Y) \\ (v.X, \bot.Y) \longmapsto \bot.f(X,Y) \\ (v.X, \mathit{false}.Y) \longmapsto \bot.f(X,Y) \\ (v.X, \mathit{true}.Y) \longmapsto v.f(X,Y) \end{cases}$$

Deterministic Merge. The SIGNAL equation $Z := X$ **Default** Y states that X and Y are merged in Z with the priority to X. **Default** is the least fixpoint of the following continuous functional:

$$F_{\mathbf{Default}}(f) =_{\text{def}} \begin{cases} (\bot.X, \bot.Y) \longmapsto \bot.f(X,Y) \\ (\bot.X, v.Y) \longmapsto v.f(X,Y) \\ (u.X, \bot.Y) \longmapsto u.f(X,Y) \\ (u.X, v.Y) \longmapsto u.f(X,Y) \end{cases}$$

Delay. The SIGNAL function **Pre** is used to access to the previous value of a signal. **Pre** is the least fixpoint of the following continuous functional:

$$F_{\mathbf{Pre}}(f) =_{\text{def}} \begin{cases} (u, \bot.X) \longmapsto \bot.f(u,X) \\ (u, v.X) \longmapsto u.f(v,X) \end{cases}$$

Using the previously defined denotations of primitive processes, we can derive the denotations of the derived operators of SIGNAL. The parallel composition is denoted by the logical *and* of the underlying logic and the introduction of local signals is denoted by an existential quantifier.

This co-inductive trace semantics of SIGNAL has been implemented with the proof assistant COQ (see [12] for details). Many lemmas are proved to ease the correctness proof of a reactive system specified with SIGNAL. The case study introduced in this paper confirms that our co-inductive approach is a natural, simple and efficient way to prove correctness of reactive systems.

3 Steam-Boiler Control Specification Problem

In order to compare the strengths and weaknesses of different design formalisms for reactive systems, the steam-boiler case study has been suggested by J.-R. Abrial, E. Börger and H. Langmaack. In this section, we briefly recall its original specification (see [2] for more details), and the additional precisions we bring (see [11] for more details).

3.1 Physical Environment

The physical environment is composed of several units (Fig. 1). Each one is characterized by physical constants and some of them provide data.

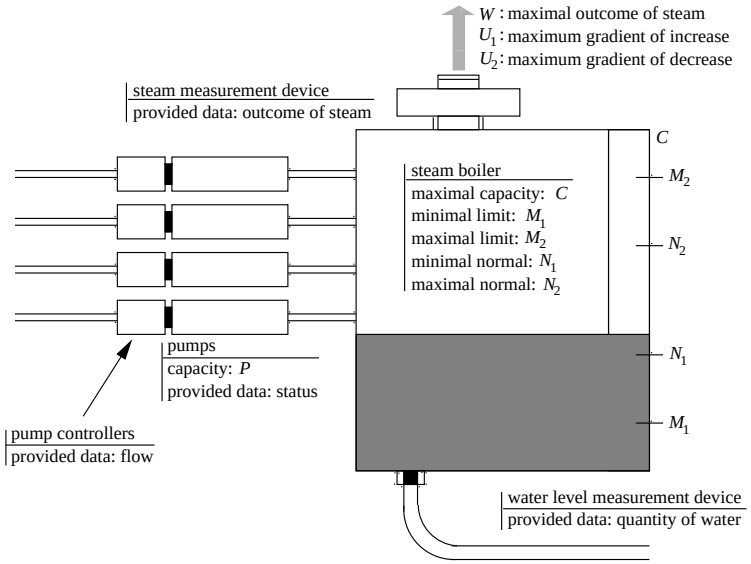

Figure1. Physical environment

3.2 Behaviour of the Steam-Boiler

The program has to control the level of water in the steam-boiler. This quantity should not be too low or too high. Otherwise, the system might be affected. The program also has to manage the possible failure of physical units. For that purpose, at every instant, it takes into account the global state of the physical environment which is denoted by an *operation mode*. The program follows a *cycle* which takes place each five seconds. A cycle consists of the reception of

messages coming from the units, the analysis of the received informations, and the transmission of messages to the units. According to the operation mode, the program decides at each cycle if the system must stop or not. If not, it activates or deactivates pumps in order to keep the level of water in the middle of the steam-boiler.

The specification also gives additional information regarding the physical behaviour of the steam-boiler. Namely, new values, called adjusted and calculated values, are proposed. They enable a sustained control of the system, by providing a vision of its dynamic, when a measurement device is defective.

At each cycle, adjusted variables contain either real measurements or extrapolated values which are calculated during the preceding cycle. An adjusted variable contains a real measurement when the corresponding device works properly. Otherwise, it contains an extrapolated value.

Calculated variables provide, at each cycle, extrapolated values of measurements for the following cycle. They contain the extreme values that are possibly reachable from the current adjusted values.

3.3 Precisions and Decisions about the Original Specification

Because of the flexibility with which the original specification of the steam-boiler can be interpreted, we first need to make some details more precise, on the physical behaviour of the steam-boiler, and on the logical behaviour of its implementation in SIGNAL. Different items are concerned by our decisions. Namely:

Distinction between pump failures and pump controller failures. We cannot rely on the fact that controllers always provide a reliable information about their associated pumps. Indeed, according to the specification, failures of controllers have to be taken into account and thus, we have to consider them as being fallible. Consequently, how could pump failures and pump controller failures be distinguished ?

We first could try to detect the real throughput of each pump with an analysis of water-level variations in the boiler. But such a method presupposes a too restrictive set of conditions about the physical characteristics of pumps and their controllers. Moreover, it actually makes controllers useless.

We have therefore chosen to determinate what the real state of each pump and controller should be, for each possible combination of values. This solution, which seems to be the most reasonable and intuitive one, was proposed in [6] (a solution of the steam-boiler problem in LUSTRE).

Message occurrences. In order to have more flexibility for controlling the steam-boiler, each pump and each controller is connected to the main program by its own communication line. Thus, each pump can be managed simultaneously and independently.

Moreover, some incoming messages from pumps essential are not always relevant for the system at every instant. For example, a pump should not necessarily provide its state if it did not receive a command during the preceding cycle. But

it can still provide its state at each cycle as specified in the original text. Only the presence of compulsory messages will be checked.

In addition to these messages, we introduce a new message H. This message is a pure signal and stands for the main clock of the program. All involved signals in the program have a clock which is a sub-clock of H. This signal is supposed to be reliable. It enables to detect the absence of compulsory messages.

Activation, deactivation of the pumps, and stop of the system. The decisions concerning the activation or the deactivation of the pumps, and the decision of stopping the system, are made according to the adjusted and calculated values. At first, a specific decision is made for each pair of extremum level, adjusted and calculated. Then, the program globally decides if the system shall stop or not. If not, the program decides how the level shall move (up or down), if necessary, and by taking into account each specific decision.

We calculate the best quantity of water to be provided, rather than just opening or closing all the pumps. Thus, at each cycle, the program calculates the optimal combination of open and closed pumps, in order to have an optimal progression of the level of water toward the middle of the boiler, taking into account failures of pumps and controllers.

3.4 Design and Architecture

The steam-boiler controller in SIGNAL is composed of four main processes (fig. 2).

- The IO_MANAGER process detects transmission failures. It implements a filter that guarantees the presence of the outgoing data, necessary to the processing. This process also provides a signal which announces the manual stop of the system.

- The FAILURE_MANAGER process is in charge of managing the dialogue between the physical units and the program regarding failure detections and repair indications. It detects failures and provides a global vision of the state of the physical system.

- The DYNAMIC process directly implements the equations suggested in the specification according to the detected failures and the values provided by the measurement devices.

- The CONTROL process is the main program. Starting from the global vision of the state of the system, and from the adjusted measurements provided by the preceding process, it manages the operation modes, makes the decision to stop the system or not, and finally delivers activation and deactivation commands to the pumps.

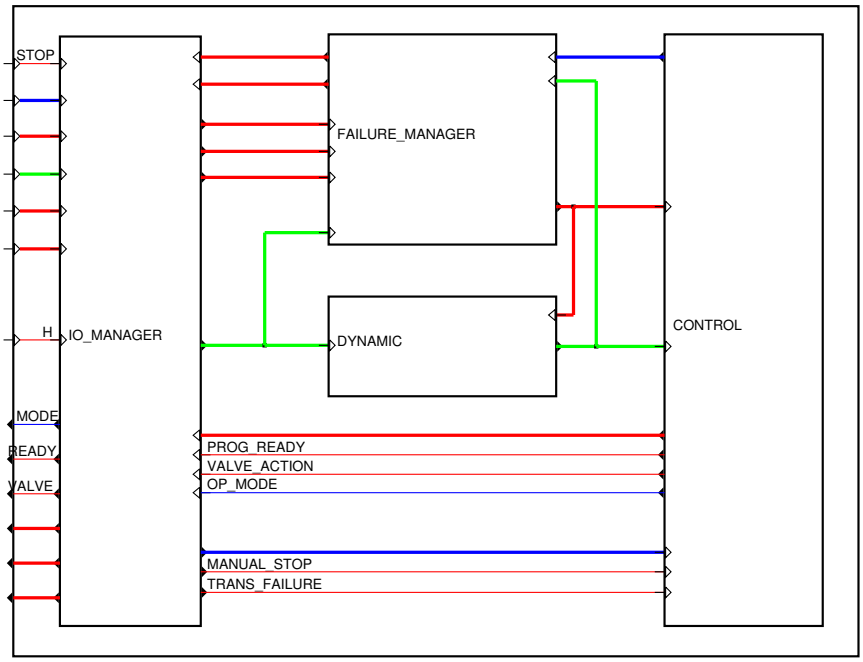

Figure2. SIGNAL processes of the steam-boiler controller

3.5 Motivation for the choice of this case study

This case study is well adapted to our aim, i.e. to show the interest of the SIGNAL-COQ formal approach. Indeed, the program has to handle several physical parameters and it may use non linear numerical values (e.g. extrapolated values of the level which take into account gradients of increase and decrease of the steam throughput, i.e. typically non linear numerical terms). Thus, safety properties cannot be simply and directly proved with a standard model checker.

4 Verification of the Steam-Boiler with Coq

Proofs of program properties are built on the co-inductive trace semantics of SIGNAL which has been implemented with COQ [13].

This axiomatization is a set of COQ libraries which gathers the modeling of signals, the modeling of the primitives of the kernel language, and a number of functions, predicates and theorems about signals. These COQ libraries, as well as the proofs of the properties that are stated in the rest of this article, are available at [12].

4.1 Safety obligations

A global safety property can be informally stated in the following way:

When a stop condition is satisfied, the system stops indeed, i.e. the program enters the emergency stop mode.

This statement implies several sub-properties. Our aim is to emphasize the interest of using Coq for their verification. Thus, in the sequel of this article, we concentrate our work especially on safety sub-properties that cannot be directly and simply proved by a standard model checker.

Since four stop conditions are specified, the global safety property has to be proved for each one:

1. *Manual stop.* The program received consecutively the required number of STOP messages from the user for manually stopping the system.

2. *Critical level.* The system is in danger because the water level is either too low or too high.

3. *Transmission failure.* The program detected a transmission failure.

4. *Initialization.* The water level measurement device is defective in the initialization mode.

In our Signal specification, these situations are associated with critical messages. When one of these signals carries a value, the corresponding condition holds and so, the program must stop.

First of all, the expected relations between these critical messages and the operation mode have to be checked. Then, we have to verify that each critical message is actually present when the condition to which it corresponds holds. For that purpose, the implied sub-properties are divided into two main classes:

- A first class gathers properties that specify the correct behaviour of critical messages, regarding the critical situations to which they correspond.
- The second class gathers properties that justify some simplifications or specify the use of some internal signals in the processing.

We now only consider sub-properties coming from *Manual Stop* and *Critical level* because they involve parameters and non linear numerical values, unlike *Transmission failure* and *Initialization*. So, they are convincing examples to illustrate our approach. Moreover, *Critical level* gathers essential properties of our solution.

4.2 Manual Stop

The problem of manual stop is generalized, using a parameter called *nb_stop*, which stands for the number of STOP messages required for manually stopping the program, instead of the fixed value "3" initially suggested in the specification.

Since we are using a proof assistant, we do not need to instantiate this parameter with a particular value. First, A predicate that denotes the right behaviour of a counter of the successive synchronous instants between two signals is co-inductively defined in Coq. Then, we prove that our Signal process provides indeed a signal (called CPT) that behaves like a counter of the successive synchronous instants between the stop signal (well called ... STOP) and the main clock (called H).

Instead of using a co-inductive predicate that denotes the expected behaviour of CPT, we define a co-recursive function that specifies CPT. This function is the least fixpoint of the following continuous functional:

$$
\begin{array}{l}
F : (\mathbb{N} \times \mathcal{F}_{(U \cup \{\bot\})} \times \mathcal{F}_{(V \cup \{\bot\})} \to \mathcal{F}_{(\mathbb{N} \cup \{\bot\})}) \\
\quad \longrightarrow (\mathbb{N} \times \mathcal{F}_{(U \cup \{\bot\})} \times \mathcal{F}_{(V \cup \{\bot\})} \to \mathcal{F}_{(\mathbb{N} \cup \{\bot\})}) \\
\quad f \longmapsto \begin{cases}
(n, \mathtt{Cons}(\bot, X), \mathtt{Cons}(\bot, Y)) \mapsto \mathtt{Cons}(\bot, f(n, X, Y)) \\
(n, \mathtt{Cons}(x, X), \mathtt{Cons}(\bot, Y)) \mapsto \mathtt{Cons}(0, f(0, X, Y)) \\
(n, \mathtt{Cons}(\bot, X), \mathtt{Cons}(y, Y)) \mapsto \mathtt{Cons}(0, f(0, X, Y)) \\
(n, \mathtt{Cons}(x, X), \mathtt{Cons}(y, Y)) \mapsto \mathtt{Cons}(n+1, f(n+1, X, Y))
\end{cases}
\end{array}
$$

Let *cssm*, the least fixpoint of F. The Coq definition of *cssm* is the following:

```
CoFixpoint cssm :
(U,V:Set)nat->(Signal U)->(Signal V)->(Signal nat) :=
[U,V:Set][n:nat][X:(Signal U)][Y:(Signal V)]Cases X Y of
 (Cons absent X')       (Cons absent Y')
  => (Cons (absent nat)    (cssm n X' Y'))
|(Cons (present _) X')  (Cons absent Y')
  => (Cons (present 0)     (cssm 0 X' Y'))
|(Cons absent X')       (Cons (present _) Y')
  => (Cons (present 0)     (cssm 0 X' Y'))
|(Cons (present _) X')  (Cons (present _) Y')
  => (Cons (present (S n)) (cssm (S n) X' Y'))
end.
```

Using this function, the predicate that denotes the expected behaviour of CPT can now be stated:

$$\mathsf{CPT} = cssm(0, \mathsf{STOP}, \mathsf{H})$$

We open in Coq a *section* in which hypotheses are stated. Those hypotheses correspond to the Signal equations which are concerned by the property to be proved:

```
(|
 ...
 | CPT ^= H
 | CPT := ((ZCPT+1) when STOP) default (0 when H)
 | ZCPT := CPT$1 init 0
 | MANUAL_STOP := when (CPT=nb_stop)
 ...
 |)
```

Those equations use constant signals. We first have to define them explicitly. Then, we have to state the hypothesis regarding H, the main clock of the program. In particular, the clock of STOP is a sub-clock of H.

This yields to the following equations:

```
0 | STOP ^< H
1 | CPT ^= H
2 | Cst0 := 0
3 | Cst0 ^= H
4 | CPT := ((ZCPT+1) when STOP) default (Cst0 when H)
5 | ZCPT := CPT$1 init 0
6 | A := (CPT=nb_stop)
7 | Csttrue := true
8 | Csttrue ^= A
9 | MANUAL_STOP := Csttrue when A
```

Using the co-inductive axiomatization of SIGNAL in COQ [13], this system of equations is translated into the following COQ hypotheses:

```
Variable nb_stop : nat.
Variables CPT,ZCPT,Cst0 : (Signal nat).
Variables H,STOP,MANUEL_STOP,Csttrue : Clock.
Variable A : (Signal bool).

Hypothesis Equation0 : (OrderClock STOP H).
Hypothesis Equation1 : (Synchro CPT H).
Hypothesis Equation2 : (Constant 0 Cst0).
Hypothesis Equation3 : (Synchro Cst0 H).
Hypothesis Equation4 :
   CPT = (SignalAA_to_SignalA
            (default (when (fonction1 [n:nat](plus n (S 0)) ZCPT)
                           (Clock_to_Signal_bool STOP))
                     (when Cst0
                           (Clock_to_Signal_bool H))) ).
Hypothesis Equation5 : ZCPT = (pre 0 CPT).
Hypothesis Equation6 : A = (fonction1 [n:nat](beq_nat n nb_stop) CPT).
Hypothesis Equation7 : (Constant tt Csttrue).
Hypothesis Equation8 : (Synchro Csttrue A).
Hypothesis Equation9 : MANUAL_STOP = (when Csttrue A).
```

In this environment, we aim at proving the following lemma:

```
Lemma l1 : CPT = (cssm 0 H STOP).
```

This property is too general for a model-checker because of the involved *nb_stop* parameter. It is also too restrictive for an inductive proof because of the instantiated parameters (values "0") involved in the `cssm` predicate and in the SIGNAL `pre` term. A more general property must be stated with non instantiated parameters. Additional hypotheses about these formal parameters can also be stated. For that purpose, the fifth SIGNAL equation of the previous specification is preferred the following, more general, one:

```
Variable ni : nat.
Hypothesis Equation5 : ZCPT = (pre ni CPT).
```

Then, the following lemma can be proved:

```
Lemma l1b : CPT = (cssm ni H STOP).
```

In particular, the initial property is verified. This is the first part of the property. Using the same method, we also prove that MANUAL_STOP provides a value when CPT reaches the *nb_stop* value. Finally, we prove that the program enters the emergency stop mode in this case.

An important feature of the method outlined in this section is that it does not at all impact the programming style because of verification constraints. SIGNAL processes are naturally translated into COQ objects (without, e.g., any variable instantiation).

4.3 Critical water level.

The property concerning the water level can be divided into several sub-properties which correspond to the different cases of critical level. Those properties involve parameters like the boiler capacity, the extremal limits of the level, or the nominal capacity of each pump. Moreover, the processing depends on the adjusted values. Thus, those properties are parameterized and concern non linear numerical values. It is therefore not possible to verify them simply and directly with a standard model checker.

At first, a set of preliminary lemmas that justify some simplifications in the program have to be proved. For instance, the following statements allow to eliminate some cases in the processing:

$$\forall t \in \mathbb{N}, \qquad qc_1(t) < qc_2(t) \tag{1}$$

$$\forall t \in \mathbb{N}, 0 \leq qa_1(t) \leq qa_2(t) \leq C \tag{2}$$

where $qa_1(t)$ and $qa_2(t)$ (resp. $qc_1(t)$ and $qc_2(t)$) stand for the minimal and maximal adjusted (resp. calculated) values of the level at instant t, and where C stands for the maximal capacity of the boiler. Indeed, the process in charge of making a decision about activations of the pumps relies on a list of the different possible interleavings of extrapolated and adjusted levels. But some of them are omitted because of the statements (1) and (2). So they have to be proved.

The adjusted values $qa_1(t)$ and $qa_2(t)$ depend on calculated values $qc_1(t)$ and $qc_2(t)$, which are defined as follows:

$$\forall t \in \mathbb{N}^*, qc_1(t) = qa_1(t-1) - va_2(t-1)\Delta t - \frac{1}{2}U_1\Delta t^2 + pa_1(t-1)\Delta t \tag{3}$$

$$\forall t \in \mathbb{N}^*, qc_2(t) = qa_2(t-1) - va_1(t-1)\Delta t + \frac{1}{2}U_2\Delta t^2 + pa_2(t-1)\Delta t \tag{4}$$

where $va_1(t)$ and $va_2(t)$ stand for the adjusted values of the outcome of steam, and $pa_1(t)$ and $pa_2(t)$ stand for the adjusted values of the cumulated throughput

of the pumps. The parameters U_1 and U_2 denote the maximum gradients of increase and decrease of the outcome of steam.

In order to prove a property equivalent to the statement (1) with a model checker, the processing would have to be changed radically. For instance, the interval of possible values could be divided into several sub-levels and then, new boolean properties about the reachability of those levels could be defined. And in every case, all parameters like U_1, U_2 or C should be instantiated. With our SIGNAL-COQ approach, we do not consider those verification problems during the design of the program. Calculated values are textually stated (cf. (3) and (4)) in SIGNAL:

```
...
| QC1 ^= QC2
| QC1 := QA1 - (VA2*Dt) - (0.5*U1*Dt*Dt) + (PA1*Dt)
| QC2 := QA2 - (VA1*Dt) + (0.5*U2*Dt*Dt) + (PA2*Dt)
| VC1 ^= VC2
| VC1 := VA1-(U2*Dt)
| VC2 := VA2+(U1*Dt)
...
```

Note that the calculated values concern the following cycle. The definition of adjusted values are naturally given from the calculated values of the preceding cycle:

```
...
| ZQC2 := QC2$1 init C
| ZQC1 := QC1$1 init 0.0
| QA2 := (Q when J_OK) default ZQC2
| QA1 := (Q when J_OK) default ZQC1
| ZVC2 := VC2$1 init 0.0
| ZVC1 := VC1$1 init 0.0
| VA2 := (V when U_OK) default ZVC2
| VA1 := (V when U_OK) default ZVC1
...
```

Signals `Q` and `V` carry the values coming from the measurement devices. Signals `J_OK` and `U_OK` provide at each cycle a boolean information about the physical state of the measurement devices. We just have to translate these SIGNAL equations into COQ hypotheses and we prove the properties (1) and (2) using co-induction. COQ offers a natural syntax for manipulating such numerical objects. For instance, consider the following statement:

$$(\forall x, y \in \mathbb{Z})(0 \leq x) \Rightarrow (0 < y) \Rightarrow (0 < x + y)$$

Using the ZArith library of COQ, the definition of this statement is the following:

```
(x,y:Z)(Zle ZERO x)->(Zlt ZERO y) -> (Zlt ZERO (Zplus x y))
```

Meanwhile, the ZArith library also provides syntactical facilities. Thus, we have an equivalent way to define this statement:

```
(x,y:Z)‘0 <= x‘->‘0 < y‘->‘0 < x+y‘
```

Such a syntax is more intuitive and so, proving equations or inequations on $\mathbb{Z}$ in Coq is much easier.

The following first lemma is very simple to prove:

```
Lemma I_N : (a,b,c,d,e:Z) ‘a <= b‘ -> ‘c <= d‘ ->
‘0 < 2*(b-Dt*c+Dt*e)+U2*(Dt*Dt) - 2*(a-Dt*d+Dt*e)-U1*(Dt*Dt)‘.
```

And then it is used to prove the statement (1):

```
CoInductive Globally2 [U,V:Set;P:(Stream U)->(Stream V)->Prop] :
(Stream U)->(Stream V)->Prop :=
  globally2 : (X:(Stream U))(Y:(Stream V))(P X Y)
              ->(Globally2 P (tl X) (tl Y))->(Globally2 P X Y).
```

This Coq statement defines a co-inductive predicate which implements the "□" connector for temporal logic. Indeed, in our co-inductive semantics of Signal, we cannot handle explicit temporal indexes (see [13] for more details).

```
Definition ltSt := [X:(Signal Z)][Y:(Signal Z)]
(x,y:Z)(hd X)=(present x)->(hd Y)=(present y)->(Zlt x y).
```

This statement defines the predicate that will be applied to the `Globally2` connector.

```
Theorem QA1_lt_QA2 : (Globally2 ltSt QC1 QC2).
```

This statement is equivalent to the statement (1)

The decision concerning the stop of the system because of a critical level is founded on the adjusted levels. Using the preceding theorem, it is very simple to prove the following property:

$$\forall t \in \mathbb{N}, qa_1(t) \leq q(t) \leq qa_2(t) \tag{5}$$

where q stands for the real level in the boiler. It means that even if a measurement device is defective, the program always knows the interval of possible current levels. Moreover, the program knows the interval of possibly reachable levels for the next cycle. Regarding these intervals, we have to check that the level is never likely to reach a critical value. For instance we have:

$$\forall t \in \mathbb{N}, (qa_1(t) \leq M_1 \wedge qc_1(t) \leq M_1) \Rightarrow \mathsf{Critical\_Level}(t) = T \tag{6}$$

It means that the program will stop (the critical message Critical_Level carries a value T) if the minimal next level is below M_1 (which is the minimal level under which the system is in danger after one cycle) while the current level is possibly already below M_1.
We also prove properties like the following one:

$$\forall t \in \mathbb{N}, (qc_1(t) \leq M_1 \wedge qc_2(t) \geq M_2) \Rightarrow \mathsf{Critical\_Level}(t) = T \qquad (7)$$

It means that if the interval of possibly reachable levels for the next cycle is too wide for making a safe decision, the program stops.

These examples emphasize an important advantage of our approach. The statements of the expected safety properties are especially clear. Moreover, the programmer does not need to have in mind what kind of property checkable or not during the design phase. Thus, specifying, programming and verifying a problem are more natural and intuitive operations.

Unlike a model checker, a proof assistant, and more generally a theorem prover cannot provide a counter-example when the check fails. But Coq gives a strong logical framework in which the user acquires a great confidence in the conformity of the program to the specification. Moreover, if the program is erroneous, the proof progression will stop on an impossible sub-goal which is often explicit enough to understand the mistake.

Nevertheless, theorem proving is often less efficient and often more tedious than model checking. Then, even if we could check all properties with only a proof assistant like Coq, the optimal solution for verification consists in using a model checker as much as possible and in using a theorem prover when a property is out of the scope of the model checker.

5 Related Works

The steam-boiler problem has become a classical case study for testing and comparing formal methods. It has been entirely specified and proved with the B tool approach ([1]). In [6], a steam-boiler has been implemented in the synchronous data-flow language Lustre (quite similar to Signal) and verified with its model-checker Lesar that allows verification of safety properties. This approach enables to prove boolean safety properties but cannot deal with numerical and parameterized properties. In [3], the semantics of Lustre has been formalized in the theorem prover PVS but co-induction is not used to represent infinite signals. The solution proposed in the Lustre-PVS approach consists of viewing signals as infinite sequences. In this setting, a signal is represented by a function which associates any instant i (a natural number) with the value v of the signal (if it is present) or with $\perp$ (if it is absent). The declarative and equational style of Signal is similar to Lustre. However, Lustre programs always have a unique reference of logical time: they are *endochronous*. Signal specifications differ from Lustre programs in that they can be *exochronous* (i.e. they can have many references of logical time). For instance, the process `x:=1 | y:=2` does not constrain the clocks of `x` and `y` to be equal. Hence, had we used functions over

infinite sequences to represent signals, we would have faced the burden of having to manipulate several, possibly unrelated, indexes of time i.

6 Conclusion

The axiomatization of the trace semantics of SIGNAL within the proof assistant COQ offers a novel approach for the validation of reactive systems.

We demonstrate the benefits of this formal approach for specifying and verifying properties of reactive systems by considering a large-scale case study, the steam-boiler controller problem.

Disregarding any compromise between the modeling tools and the modeled system, we augmented the original specification of the steam-boiler of [2] with a more precise description of the physical environment.

This case study shows to be well adapted to the evaluation of the SIGNAL-COQ formal approach, allowing the modeling of parameterized strong safety property with non-linear numerical constraints. In spite of the strong implication for the user during the proof-checking process, it appears that the use of a proof assistant like COQ has many advantages.

In addition to the facts that the approach alleviates any limitation in the expression of properties, it makes it possible to acquire a strong confidence in the system being specified. Moreover, it is noticeable that experiences at using COQ allowed to develop libraries which improved the efficiency of latter proofs.

However, this approach is interesting only with properties that cannot be directly proved by a model checker. It is thus advisable to use a proof assistant in complement to more classical approaches to check these particular (e.g. parameterized, co-inductive, non-linear) properties. In conclusion, the integration of model-checking and theorem-proving within a unified framework seems to be a promising prospect.

References

1. J.-R. Abrial. *The B-Book*. Cambridge University Press, 1995.
2. J.-R. Abrial, E. Börger, and H. Langmaack. Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. *Lecture Notes in Computer Science*, 1165, October 1996.
3. S. Bensalem, P. Caspi, and C. Parent-Vigouroux. Handling Data-flow Programs in PVS. Research report (draft), Verimag, May 1996.
4. A. Benveniste and P. Le Guernic. Synchronous Programming with Events and Relations: the SIGNAL Language and its Semantics. *Science of Computer Programming*, 16(2):103–149, 1991.
5. G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.
6. T. Cattel and G. Duval. The Steam-Boiler Problem in LUSTRE. *Lecture Notes in Computer Science*, 1165:149–164, 1996.

7. B. Barras et al. *The Coq Proof Assistant Reference Manual - Version 6.2*. INRIA, Rocquencourt, May 1998.
8. E Giménez. *Un Calcul de Constructions Infinies et son Application à la Vérification des Systèmes Communicants*. PhD thesis, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, December 1996.
9. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language LUSTRE. *Proc. of the IEEE*, 79(9):1305–1320, September 1991.
10. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
11. M. Kerbœuf, D. Nowak, and J.-P. Talpin. The Steam-boiler Controller Problem in SIGNAL-COQ. Research Report 3773, INRIA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France), October 1999.
12. `http://www.irisa.fr/prive/Mickael.Kerboeuf/gb/SBGB.htm`.
13. D. Nowak. *Spécification et preuve de systèmes réactifs*. PhD thesis, IFSIC, Université Rennes I, October 1999.
14. D. Nowak, J.-R. Beauvais, and J.-P. Talpin. Co-inductive Axiomatization of a Synchronous Language. In *Proceedings of Theorem Proving in Higher Order Logics (TPHOLs'98)*, number 1479 in LNCS, pages 387–399. Springer Verlag, September 1998.
15. B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Université Paris VII, May 1994.